CS112 - Fall 2022

Lab17

Instructor: Paul Haskell

INTRODUCTION

After some intense lectures on advanced class design and inheritance, it's time to create an inherited class yourselves.

CSVWriter

For your first project this week, you will create a tool that I had to create a few weeks ago to help grade the Blackjack project. "CSV" stands for "comma separated values", which is the name of a format for text files that contain multiple lines, each line with the same number of entries, with each line's entries separated by commas. CSV files are used because they are "plain text", i.e. readable by humans if you just print out all the characters. But this format is also readable directly by spreadsheet programs such as Microsoft Excel. This format is an "interchange format" that allows data to be transferred between different spreadsheet programs or between spreadsheet and non-spreadsheet programs.

You will write a program called **CSVTester.java**. In this file, write a class called `CSVWriter`. Please derive `CSVWriter` from `FileWriter`. **CSVTester.java** also should contain a `class CSVTester`, with just a `main()` method, which tests `CSVWriter`.

Ok, what should class `CSVWriter` do? It should have a constructor that takes a `File` input and another that takes a `String` input—the `String` holds the pathname of the `File` to be written.

CSVWriter should implement a `void writeln(String[] stringsForALine)` method. This method uses the underlying `FileWriter write()` method to write each of the `stringsForALine` to the `File` specified in the constructor. And of course, the `CSVWriter writeln()` function should put a comma after every `String` in every line, except not after the last `String` in each line. The last `String` in each line is simply followed by a NEWLINE.

That's it? Not quite, but that's most of it. The user of `CSVWriter` will want to write multiple lines to a file. The spreadsheet programs that read the CSV files insist that every line have the same number of entries. Please create a new `Exception` class (you pick the name), derived from `class Exception`, and have your `writeln()` method throw your new exception if the user calls `writeln()` with some number of strings per line that differs from the number in all previous `writeln()` calls. The text in your new `Exception` type should give the expected number of strings per line (from earlier calls to `writeln`) and also the number in the troublesome `writeln` call.

Your **CSVTester.java** program should keep reading the **keyboard** input until the keyboard input terminates. The program should take each line of keyboard input, parse out the separate words (separated by one or more spaces), and then call `CSVWriter.writeln()` with a list of all the words for each line. Please have the program output to a file called "**CSVTester.csv**" . And as hinted above, if the user does not always enter the same number of words per line, **CSVTester.java** should fail with your

new `Exception`. (The autograder can tell that the program failed and can see the text in your exception—of course your testing should try this also.)

Does your program need any special handling for inputs in double quotes (e.g. treating "cat dog" as one word rather than two)? No! The special double-quote handling that we have talked about is a feature of your Operating System (MacOS or Windows or Linux). It is not done by Java and it is not needed from your program.

Still having fun? One last exercise. Create yet another new `Exception` type, derived from the new `Exception` type you defined previously. This can be called the `EmptyLineException`. Throw this exception if the user enters an empty line of text. Now have `class CSVTester` catch this new `Exception` type, and use it to close the `CSVWriter` and terminate the program cleanly, with proper data output and with no error messages.

To review:

- If user keeps entering same number of words per line and then terminates input, `CSVWriter` writes each line's words with comma separators
- If user changes the number of words per line, `CSVWriter` throws your new `Exception`, `CSVTester` does not catch it, and the program crashes with an error message
- But if user enters a line with zero words, even if there is more input afterwards, then `CSVWriter` throws an `EmptyLineException`, `CSVTester` catches it, closes the `CSVWriter`, and exits the program

Phew!

Reminder

Put your file in **Lab17** and push to GitHub before the deadline. This assignment must be turned in before 11:59pm Tuesday November 1st.

Conclusion

This program gives you some experience making derived classes, and in particular derived Exceptions. You also get a tool that might be useful some day: a program that formats data so it can be read directly into a spreadsheet program

Grading Rubric

CSVTester.java is worth 30 points: 4 points for each of 5 test cases.

Software design and quality is worth 10 points.